Graph Neural Network Tracking for Particle Physics

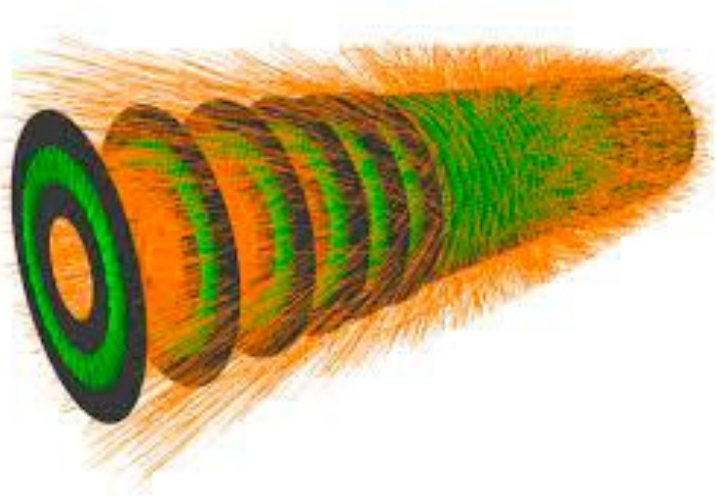
Shih-Chieh Hsu

University of Washington / A3D3

Jan 18 2024
IAS HEP 2024 Mini-workshop
Experiment and Detector

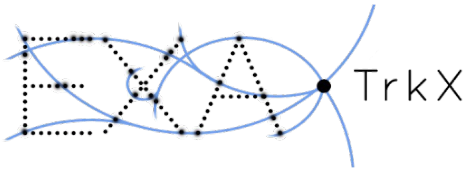
Tracking Challenges in HL-LHC

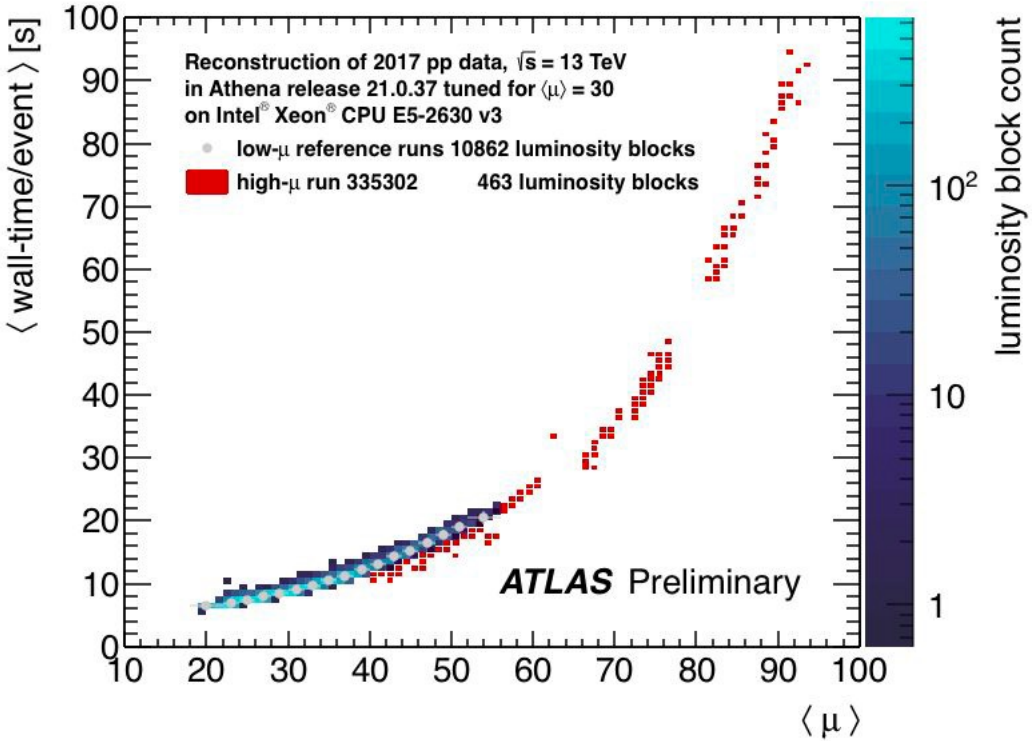
- Track reconstruction presents a challenging pattern recognition problem at the High Luminosity Large Hadron Collider (HL-LHC)
  - In High Luminosity LHC, <μ> ~ 200
  - O (10k) particles, O(100k) hits



Limitations of conventional algorithms

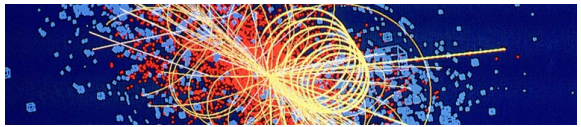
- Conventional algorithms have trouble scaling to dense environment
  - Tracking takes ~40% of total event reconstruction time
  - Cannot be easily ported to parallel devices such as GPUs
- Call for novel solutions
  - Such as deep learning methods: ExaTrkX

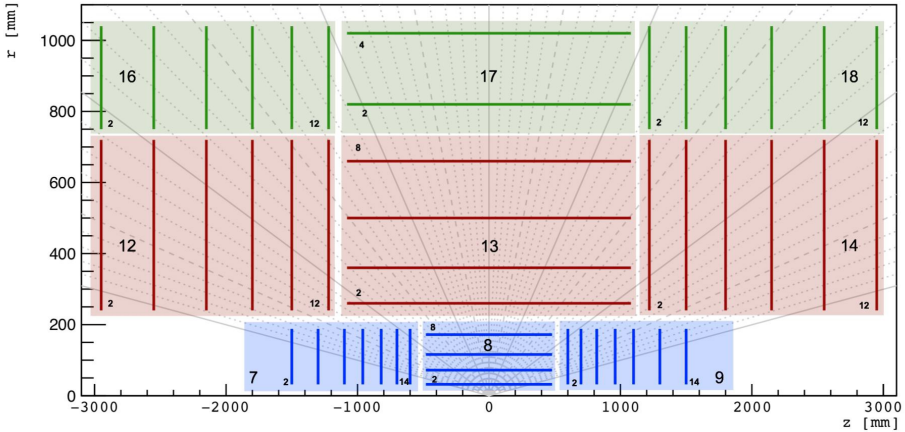


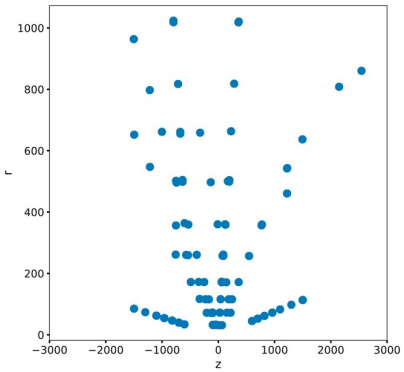


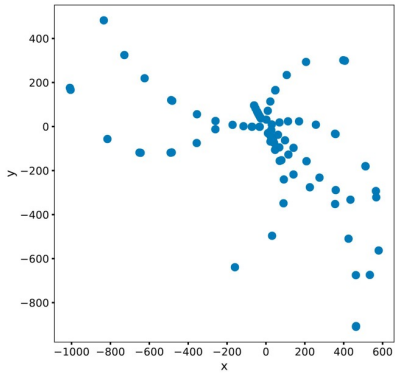
TrackML Challenges

- TrackML challenge dataset
  - https://www.kaggle.com/c/trackml-particle-identification
- Detector is highly segmented and the data size is dynamic
- The detector is emerged in a magnetic field, charged particles leave a trajectory in the detector
- The objective of machine learning is to reconstruct those trajectories



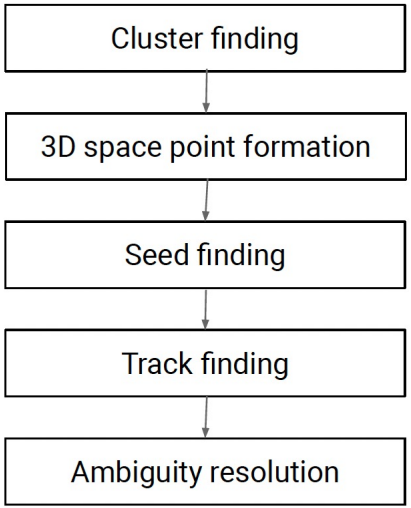


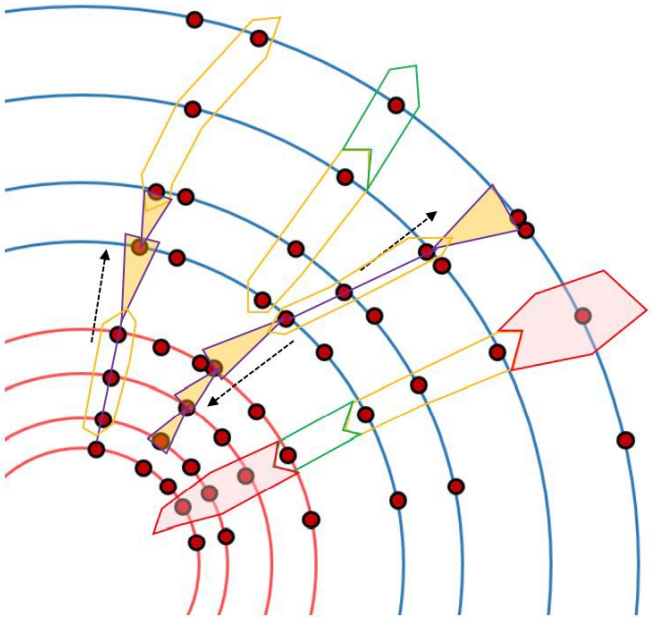




Conventional algorithm

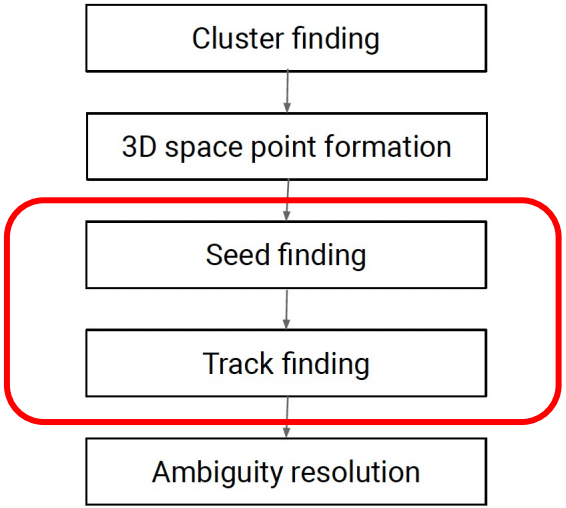
- Recursive and branching nature makes it difficult to run the conventional alg. in highly paralleled devices (such as GPUs)



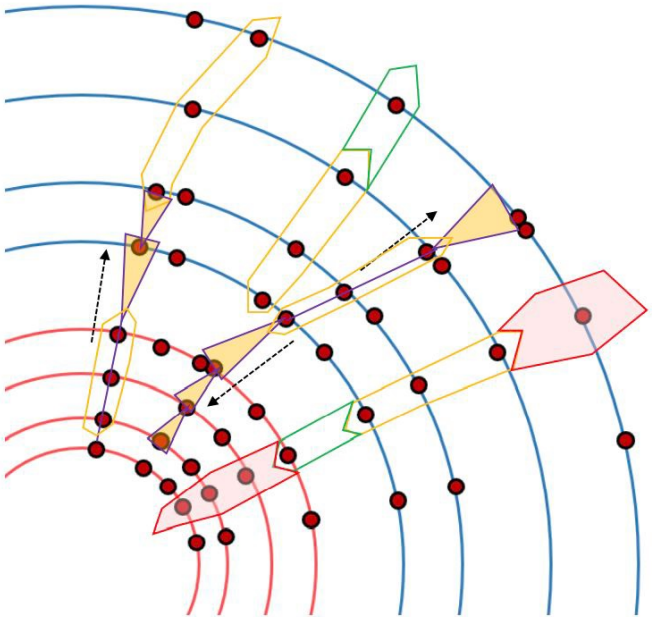


Conventional algorithm

- recursive and branching nature makes it difficult to run the conventional alg. in highly paralleled devices (such as GPUs)



Most computational expensive part

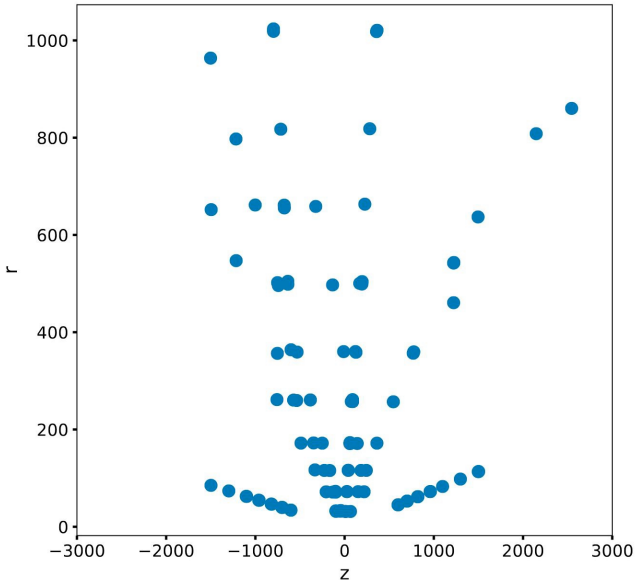


The ExaTrkX solution

- Challenges
  - Too many tracks (10,000) and dynamic number of tracks in each event
  - Each hit has high-dimensional features (>=3D)
- ExaTrkX Solution
  - Data abstraction

    → point cloud (a set of data points in the measured space)
  - Objective of ML

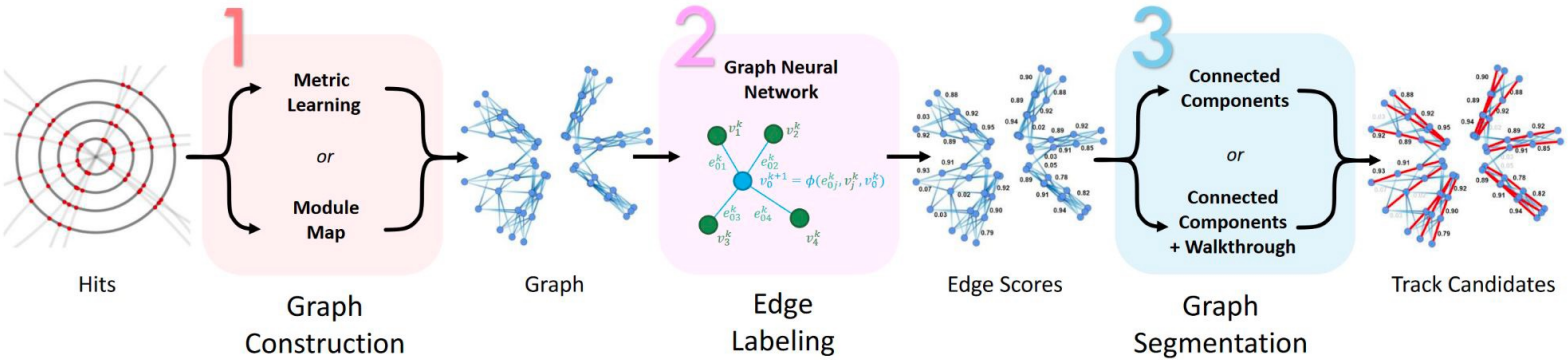
    → learn local relational information between two hits

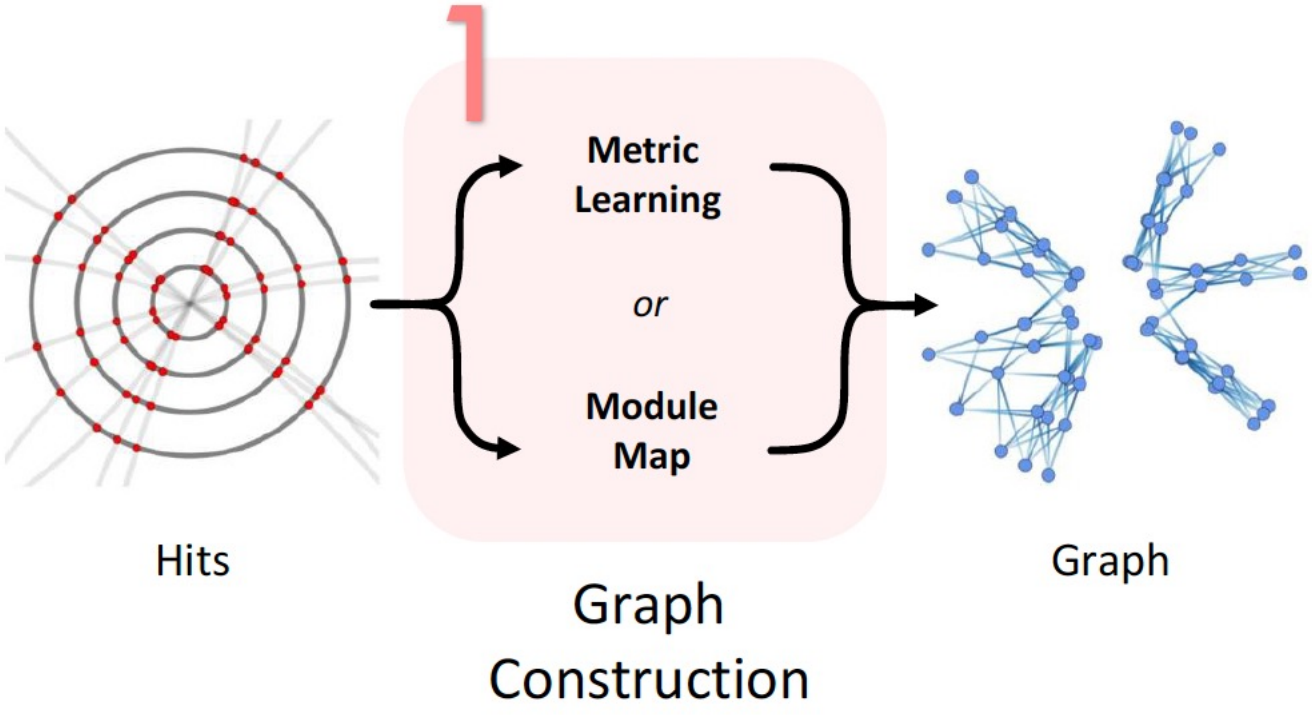


ExaTrkX pipeline

Charline Rougier, CTD 2022

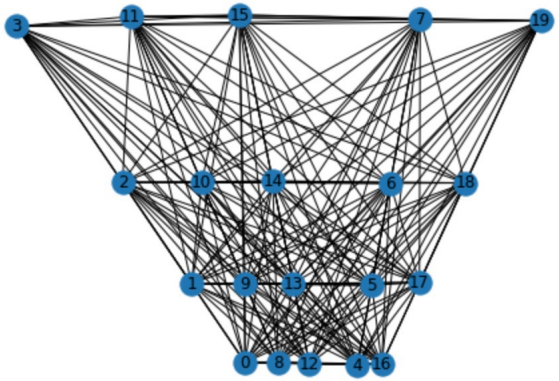
- Proof of principle by Exa.TrkX project Method applied to TrackML data by Exa.TrkX and L2IT project




1
Metric Learning
or
Module Map
Hits
Graph
Graph Construction

Graph presentation of tracking data

- Node = 1 space-point
- Edge = connection between two nodes.



Example with 19 hits in the (z,r) plane

O(300k) space-points in an event
=> fully connected graph (10^{10}) edges
=> Comprises unphysical connections

How de we choose the connections between nodes?

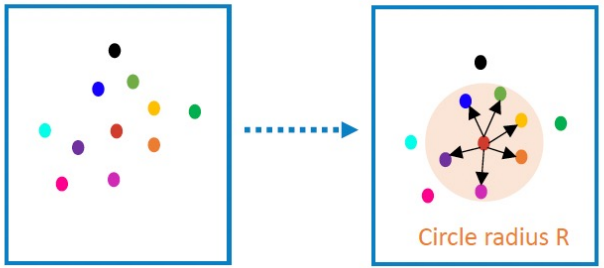
Graph creation: learning connections (two methods)

Metric Learning

Given a source node, edges between this node and all nodes within a radius R from the source are created.

N-dimensional space learned by the MLP

Edges created

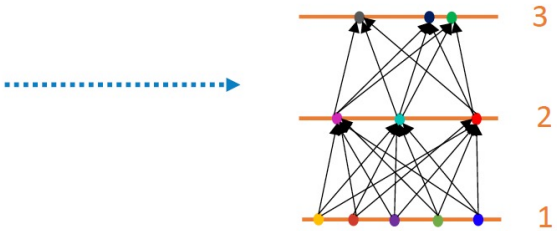


No particular meaning of direction.

Module Map

Edges are created following the connections of the Module Map.

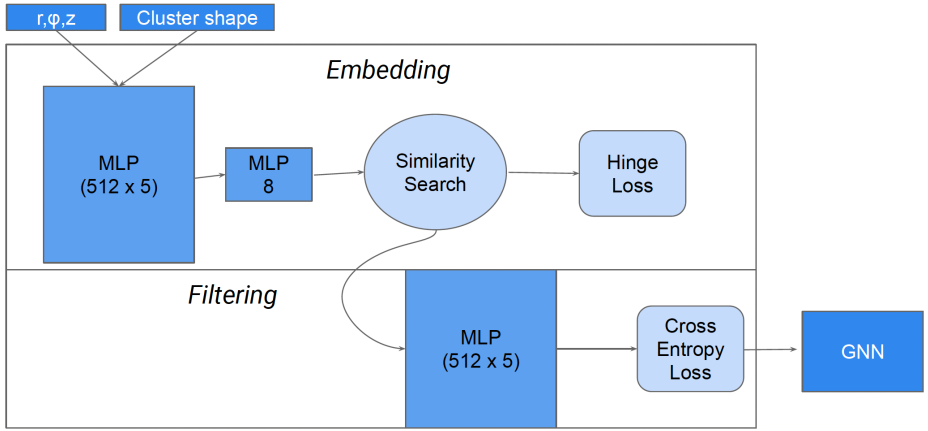
$1 \rightarrow 2 \rightarrow 3$
$2 \rightarrow 3 \rightarrow 5$
$3 \rightarrow 5 \rightarrow 6$

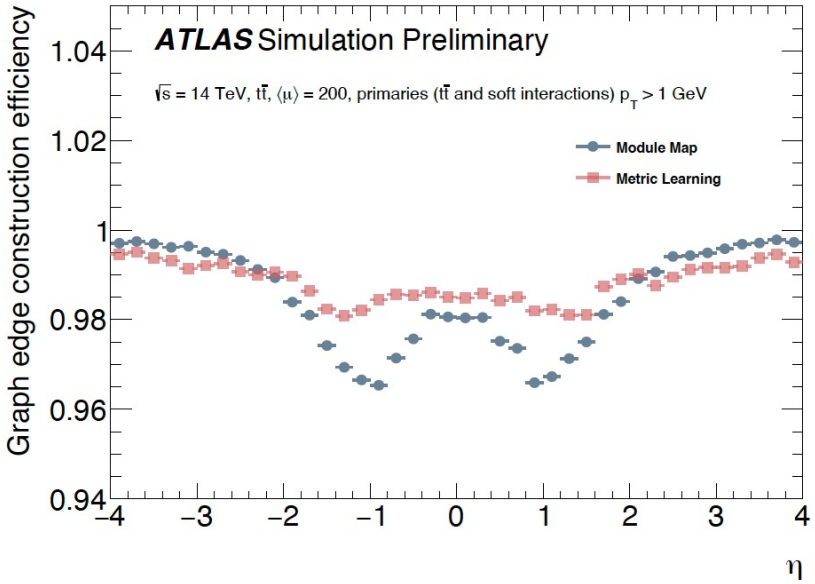


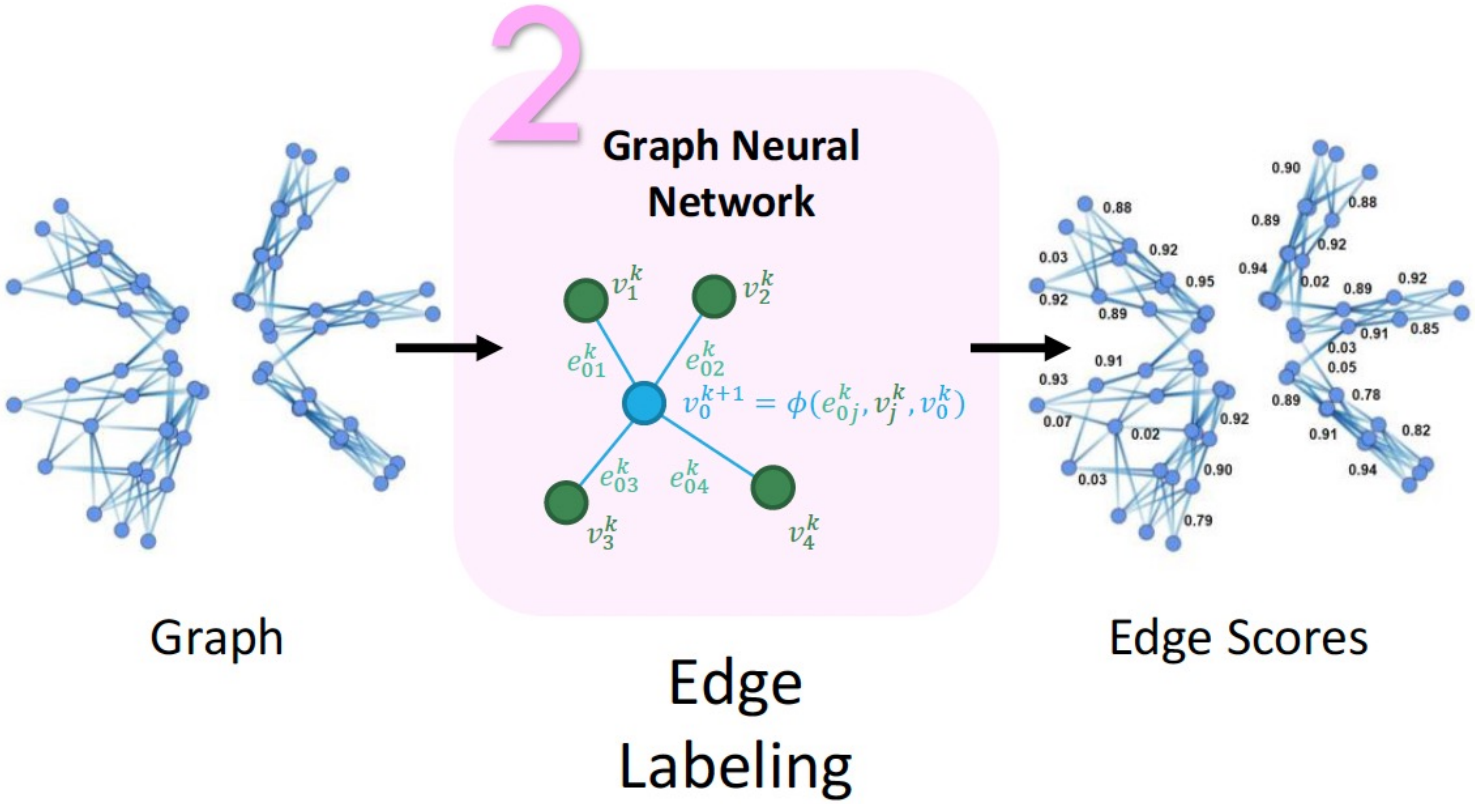
Direction "***inside-out***" are given to edges.

Graph edge construction efficiency

- High efficiency is a necessity: an edge lost during the graph construction can't be retrieved later.

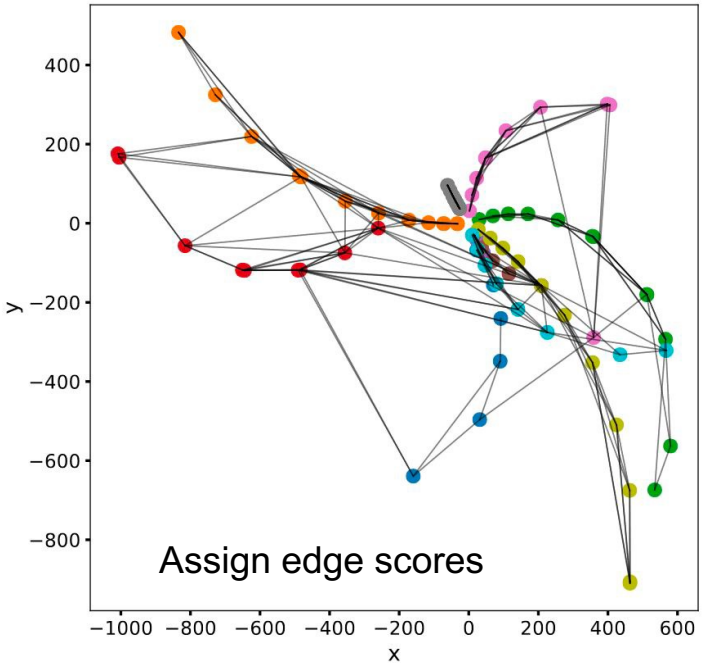




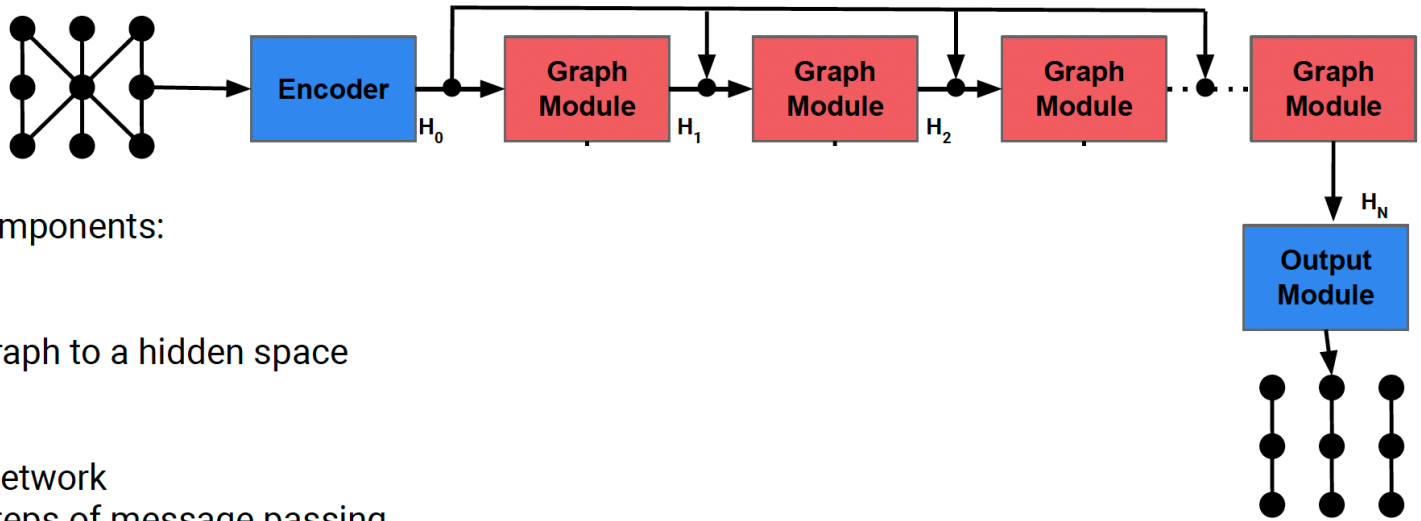

2
Graph Neural Network
v_1^k
v_2^k
e_{01}^k
e_{02}^k
$v_0^{k+1} = \phi(e_{0j}^k, v_j^k, v_0^k)$
e_{03}^k
e_{04}^k
v_3^k
v_4^k
0.88
0.03
0.92
0.95
0.92
0.89
0.91
0.93
0.07
0.02
0.92
0.03
0.90
0.79
0.90
0.89
0.88
0.94
0.92
0.02
0.89
0.92
0.91
0.85
0.03
0.05
0.89
0.78
0.91
0.82
0.94
Graph
Edge Labeling
Edge Scores

Graph Neural Network

- GNN normally composed with node, edge and graph network
  - Node / Edge network can be MLPs, Recurrent Neural Networks, or Convolutional Neural Networks when node is an image
- **Message passing**: unique component of GNN
  - allows node and edge exchange information to learn sophisticated hidden features



Edge classifier



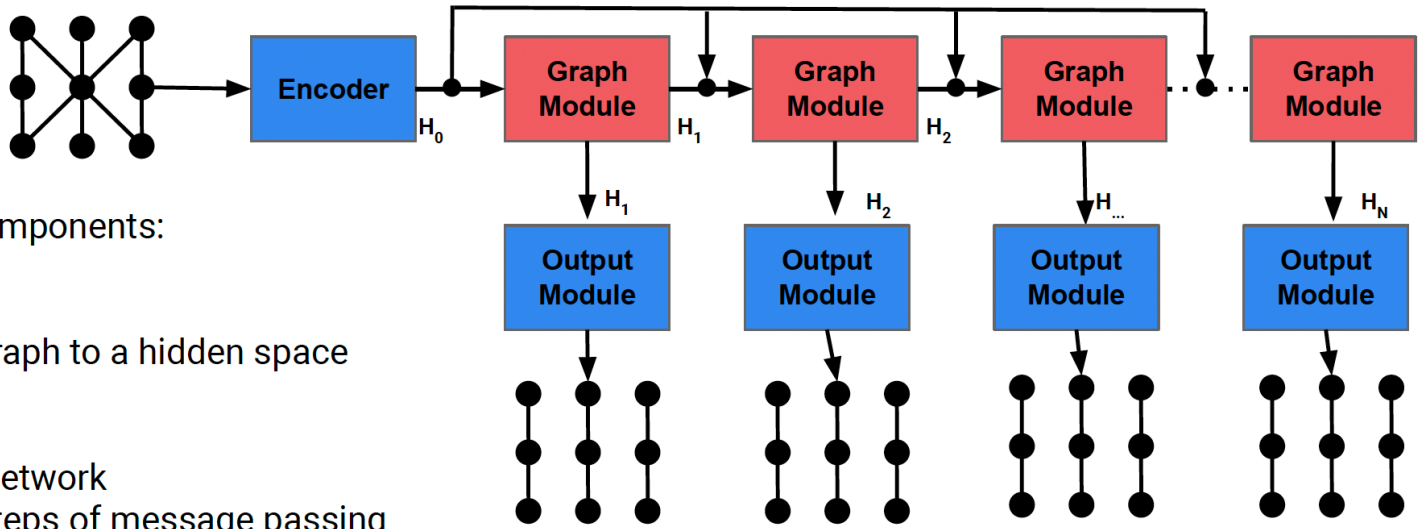
Three major components:

Encoder
→ map input graph to a hidden space

Graph Module
→ interaction network
→ perform *N* steps of message passing through the graph, *N*=8

Output Module
→ make edge predictions

Edge classifier



Three major components:

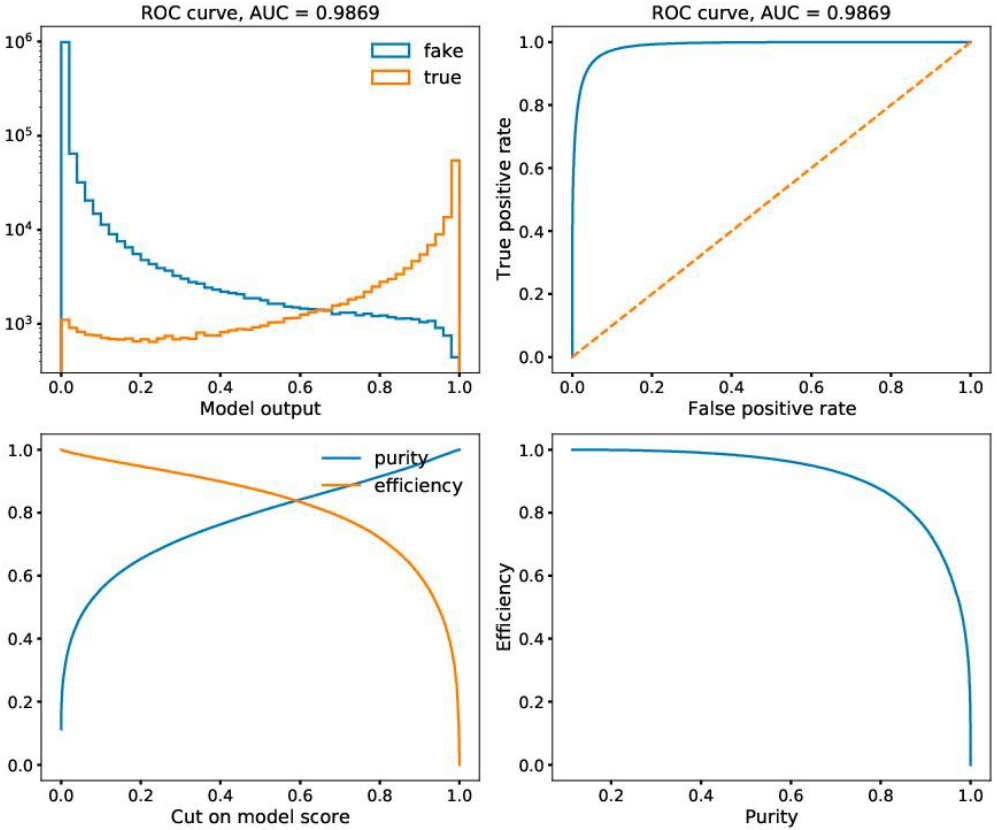
Encoder
→ map input graph to a hidden space

Graph Module
→ interaction network
→ perform *N* steps of message passing through the graph, *N*=8

Output Module
→ make edge predictions

Intermediate edge predictions are also added to the loss function.

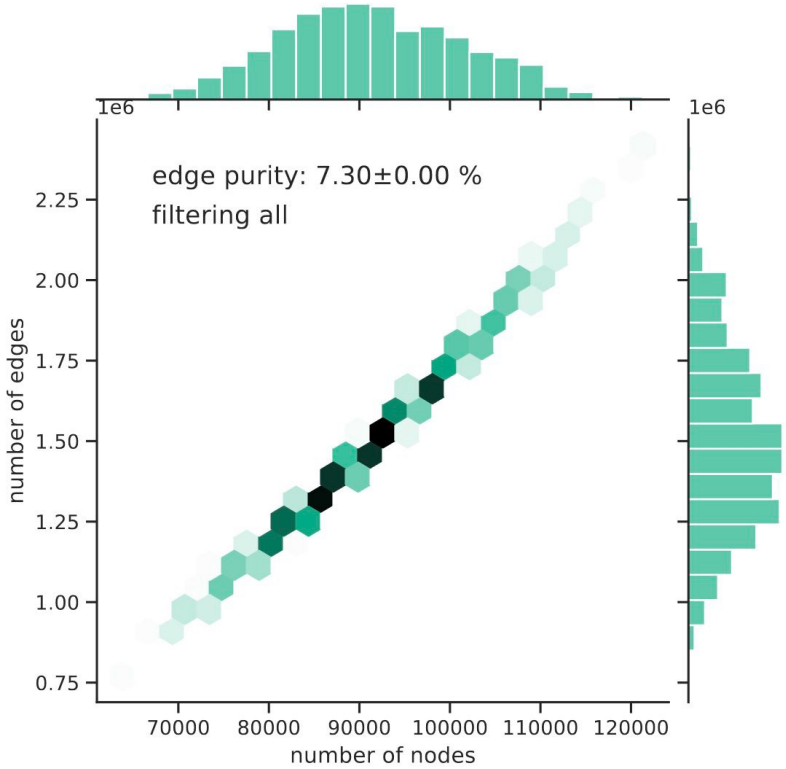
Effect of message passing

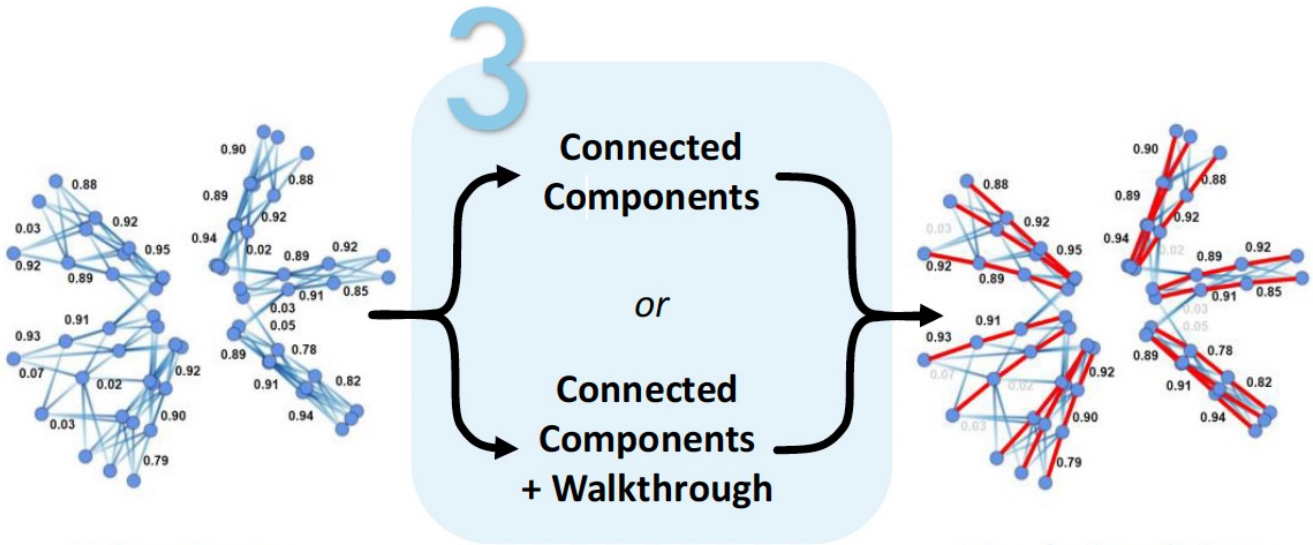


- Changes in AUC for each step: 0.9869 → 0.9972 → 0.9991 → 0.9995 → 0.9997 → 0.9997 …
- By recursively updating node and edge features, the GNN model improves its edge predictions

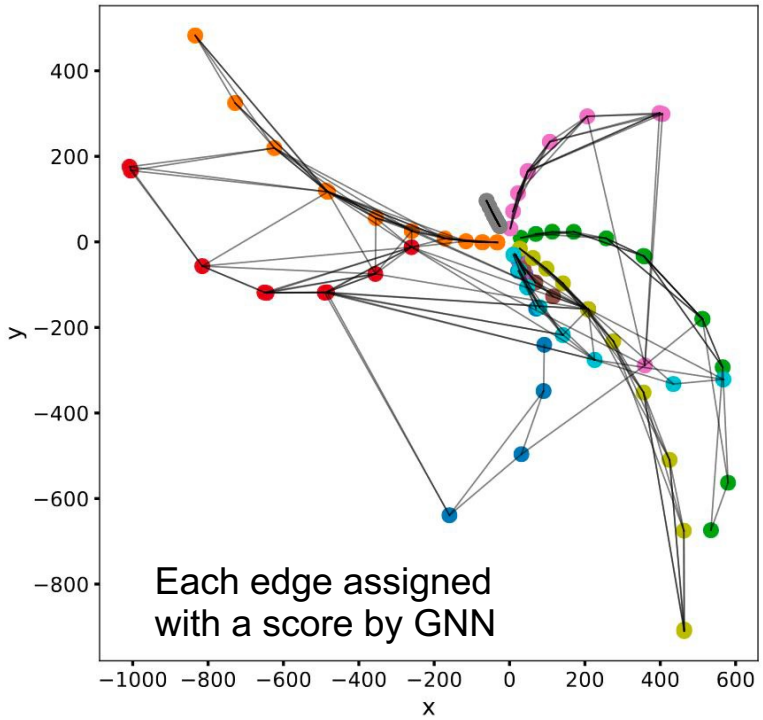
Input to GNN

- Average graph size:
  - number of nodes: 90,000
  - number of edges: 1,500,000
- Using 8500 training events to train the GNN
  - each epoch takes about 70 minutes running on NVIDIA GPU A100
  - Takes more than one day to converge a good result
- The memory consumption reaches the limitation of A100.



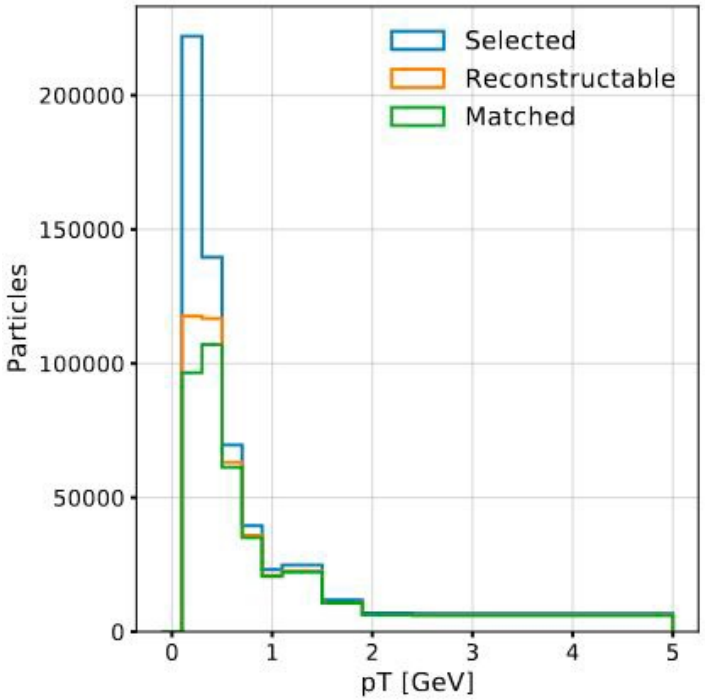

3
Connected Components
or
Connected Components + Walkthrough
Edge Scores
Graph Segmentation
Track Candidates

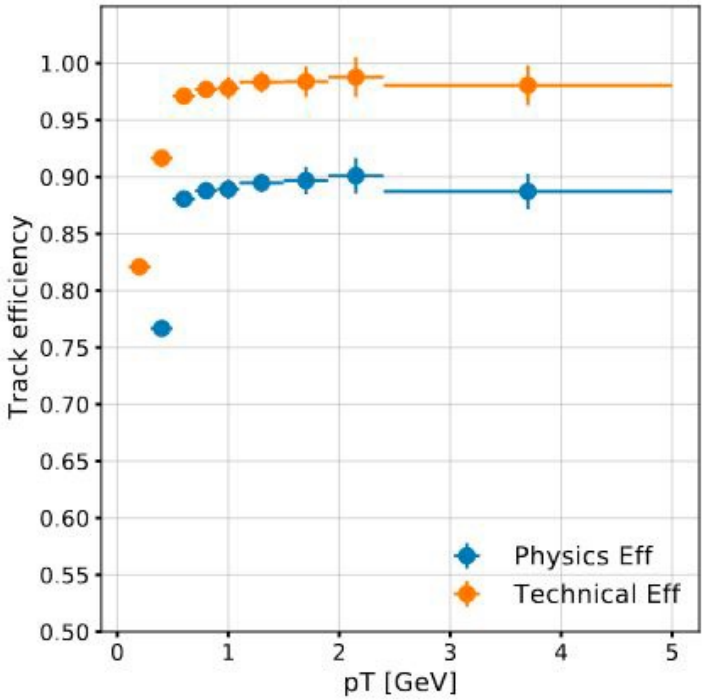
Track labeling



- The output of GNN is weighted graph
- Apply DBScan to form track candidates
- Or traditional graph algorithm cuGRAPH: weakly connected components
- The two give very similar results.
  - The later is 1 ~ 2 orders of magnitude faster than the former.

Tracking Performance



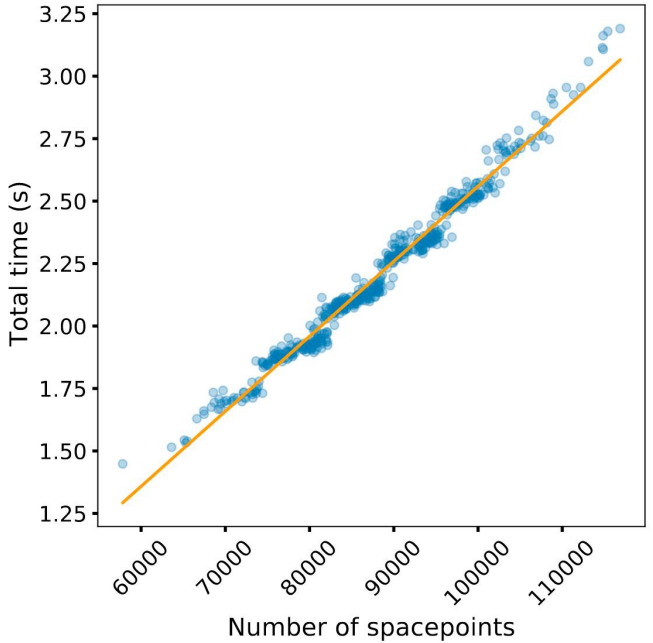


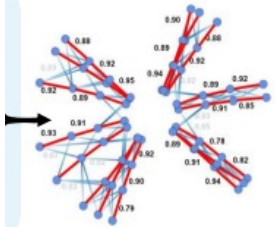
Accelerating the ExaTrkX pipeline

arxiv:2202.06929

- Total time is reduced from 15 seconds to 0.7 seconds
- Optimization of the pipeline on GPUs
  - FAISS library for nearest neighbor search
  - cuGraph, GPU implementation of connected components
  - AMP, automatic mixed precision
  - FRNN, fixed radius nearest neighbor

	Baseline	Faiss	cuGraph	AMP	FRNN
Data Loading	0.0022 ± 0.0003	0.0021 ± 0.0003	0.0023 ± 0.0003	0.0022 ± 0.0003	0.0022 ± 0.0003
Embedding	0.02 ± 0.003	0.02 ± 0.003	0.02 ± 0.003	0.0067 ± 0.0007	0.0067 ± 0.0007
Build Edges	12 ± 2.64	0.54 ± 0.07	0.53 ± 0.07	0.53 ± 0.07	0.04 ± 0.01
Filtering	0.7 ± 0.15	0.7 ± 0.15	0.7 ± 0.15	0.37 ± 0.08	0.37 ± 0.08
GNN	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03
Labeling	2.2 ± 0.3	2.1 ± 0.3	0.11 ± 0.01	0.09 ± 0.008	0.09 ± 0.008
Total time	$15 \pm 3.$	3.6 ± 0.6	1.6 ± 0.3	1.2 ± 0.2	0.7 ± 0.1





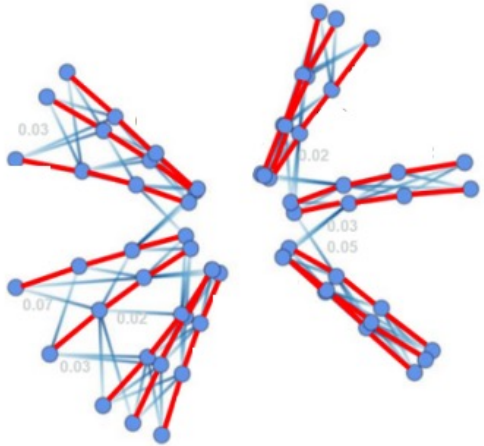
Track Candidates

4

Athena

Standard
ATLAS software

χ^2 fit
accounting for expected
multiple scattering effects



ATLAS track candidates
with track parameter
(p_T, θ, ϕ, d_0, z_0)

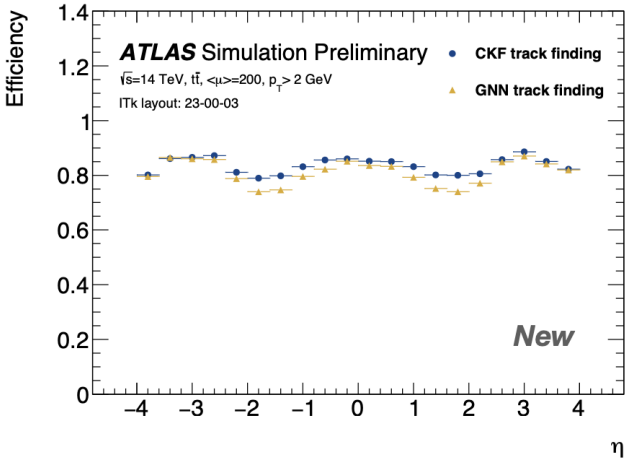
Tracking efficiency

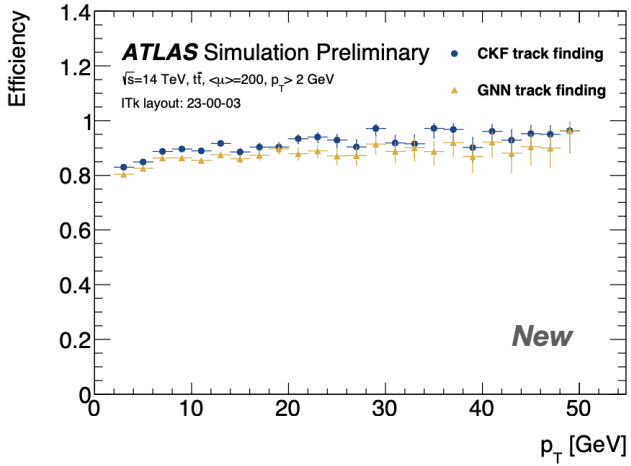
H. Torres CTD 2023

GNN4ITk, L2IT

- Competitive “physics” efficiency
  (excluding electrons)

[O(10^{-3}) fake tracks:
Track candidates not matched to any particle]

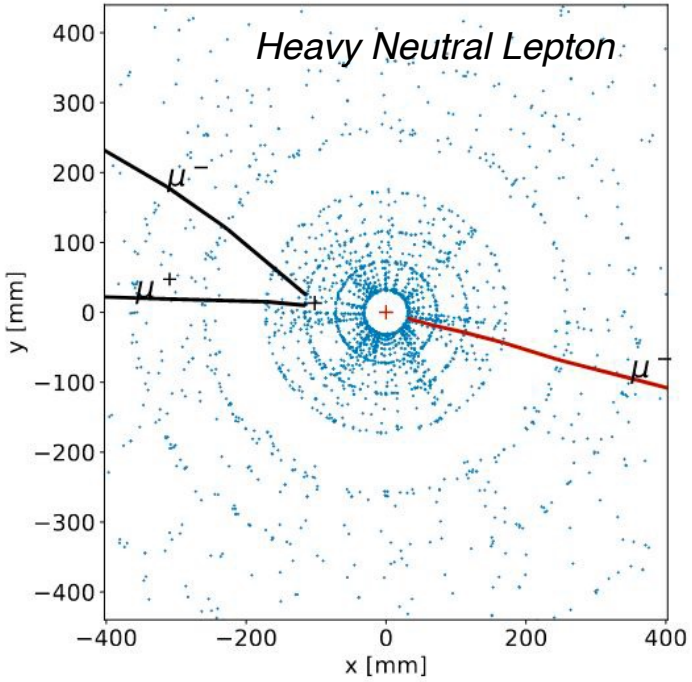


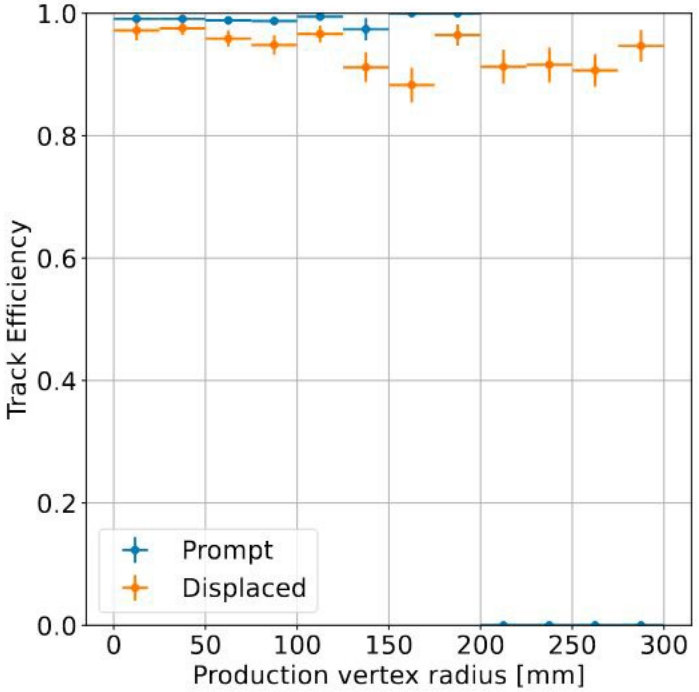


Large Radius Tracking with GNN

arxiv:2203.08800

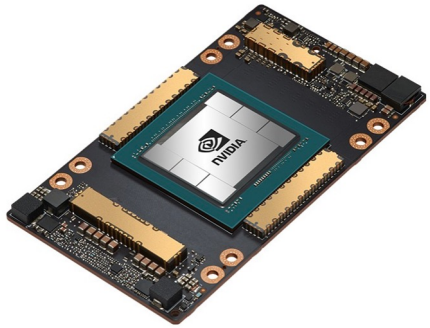
- ExaTrkX is able to reconstruct prompt and displaced tracks at the same time.

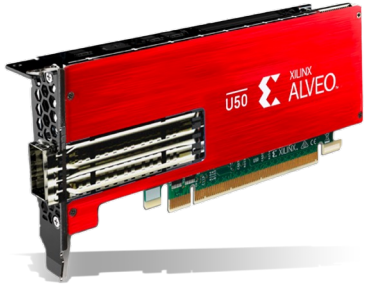




Acceleration for Tracking

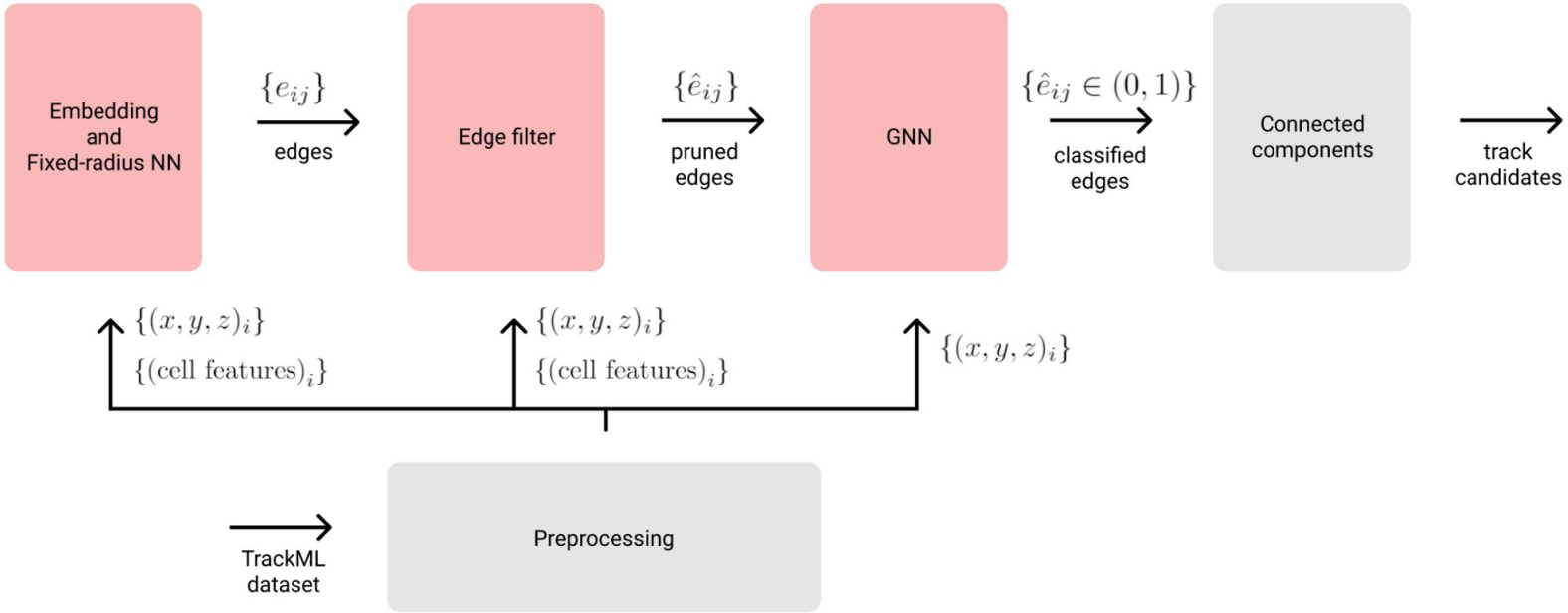
- The use of co-processor (GPU, FPGA) can potentially to accelerate latency
- Heterogeneous computing workflow required additional care to improve throughput



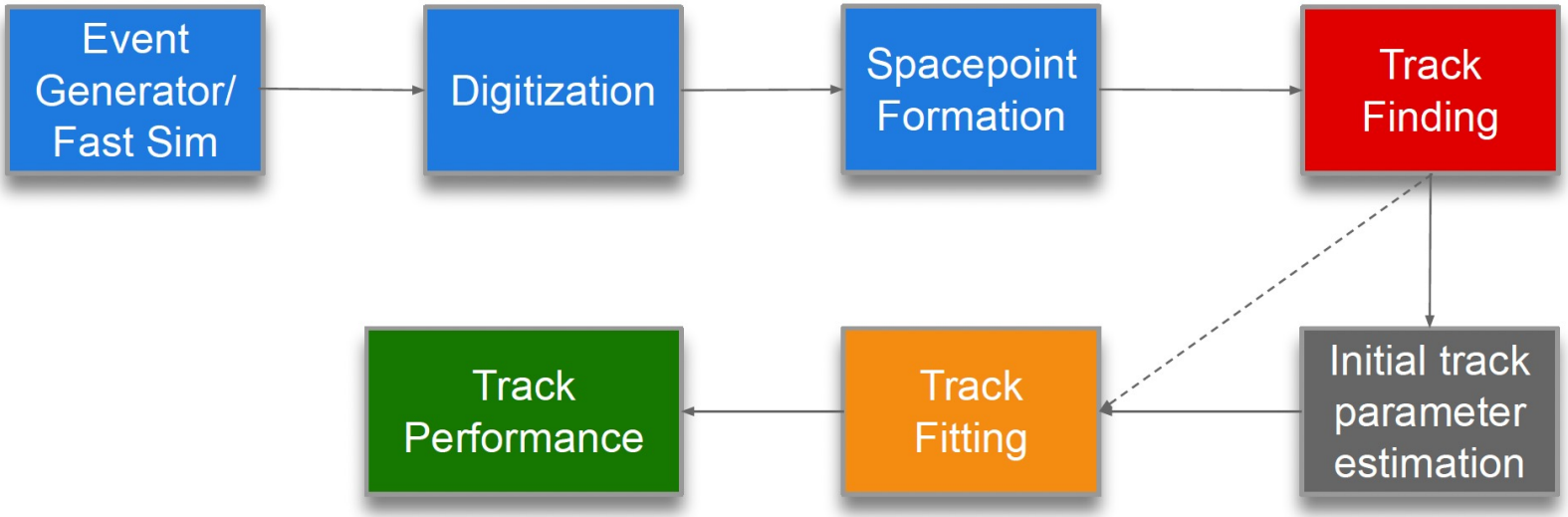


How can we enhance the scalability of the tracking workflow and make the integration much easier?

ExaTrkX track finding pipeline



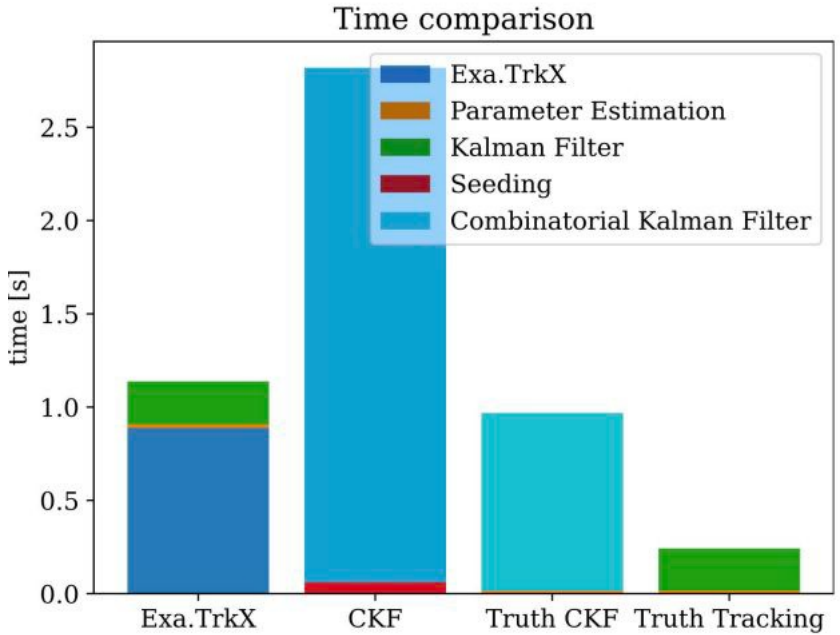
Integration into production framework



- Use the ExaTrkX pipeline for Track finding
- Integrated into the A Common Tracking Software (ACTS)

Combinatoric KalmanFilter vs ExaTrkX

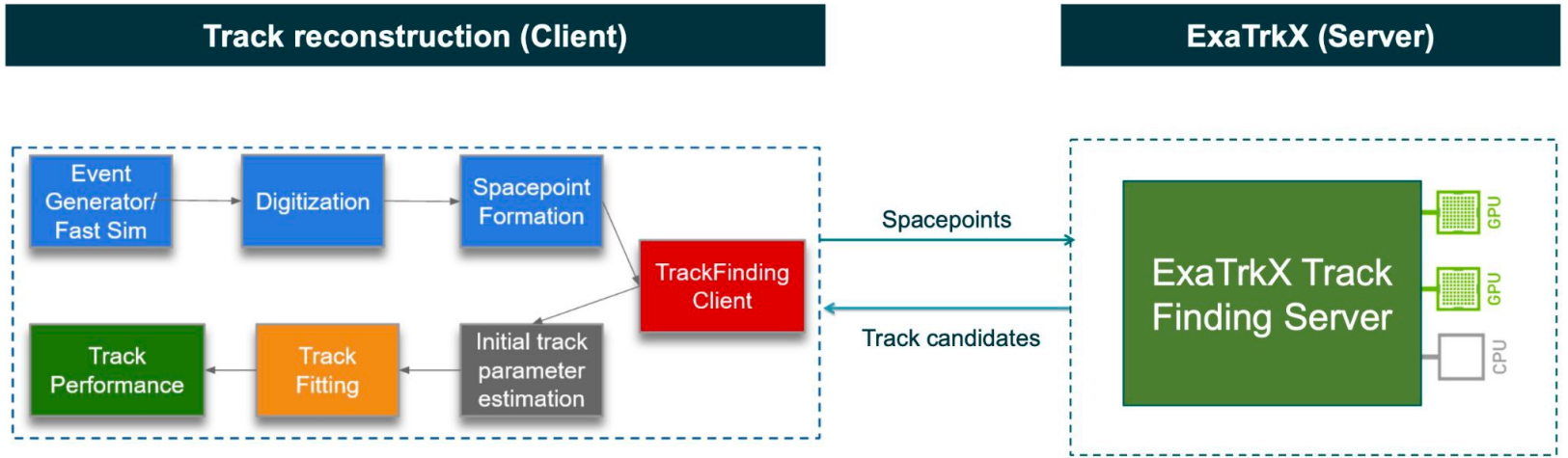
B. Huth CTD 2023



- Integrating into ACTS allows us to compare ExaTrkX pipeline with the existing algorithms
- A preliminary computing time comparison between conventional algorithms (CKF) and the ExaTrkX
- ExaTrkX was run in GPUs, while CKF in CPUs
- A GPU-version of ACTS is under development [traccc]. Would be interesting to compare ExaTrkX with the GPU version.

ExaTrkX as a service

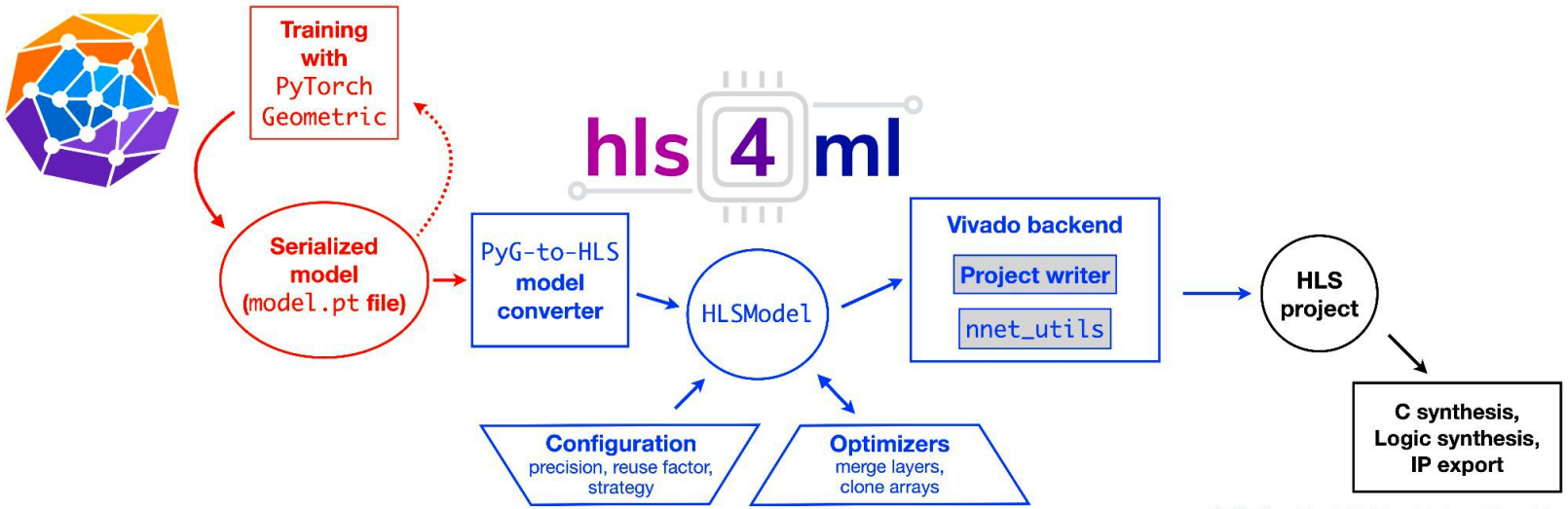
- Run the ExaTrkX track finding algorithm as a server
- Clients send requests to the server in order to get track candidates
- Standalone implementation: https://github.com/exatrkx/exatrxk-cpp-ctd2022



GNNs for Particle-Tracking on FPGAs

Front. Big Data 5:828666 (2022)

- Throughput-optimized: small graphs (28 nodes, 56 edges), < 1us latency
- Resource-optimized: large graphs (1344 nodes with 2688 edges), 6 us latency



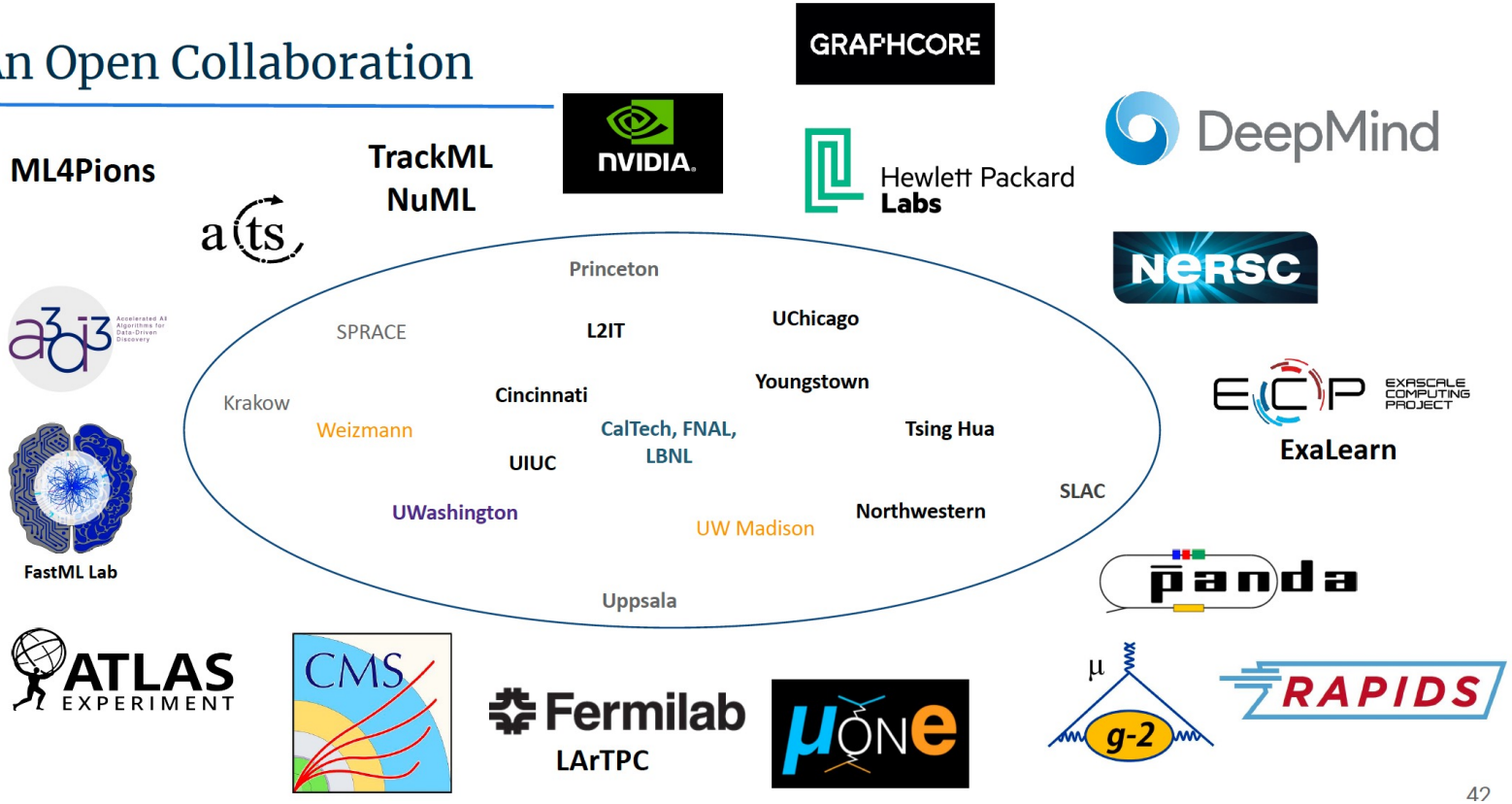
XILINX

Summary

- GNN-based track reconstruction with simulated data are promising and realistic.
  - TrackML proof of principle, GNN4ITk provides competitive performance
- Prospect from active development
  - Full Integration to ACTS
  - Fair comparison to CKF GPU
  - Optimized latency and throughput through hardware-algorithm co-development and ML-aaS heterogenous computing

Backup

An Open Collaboration

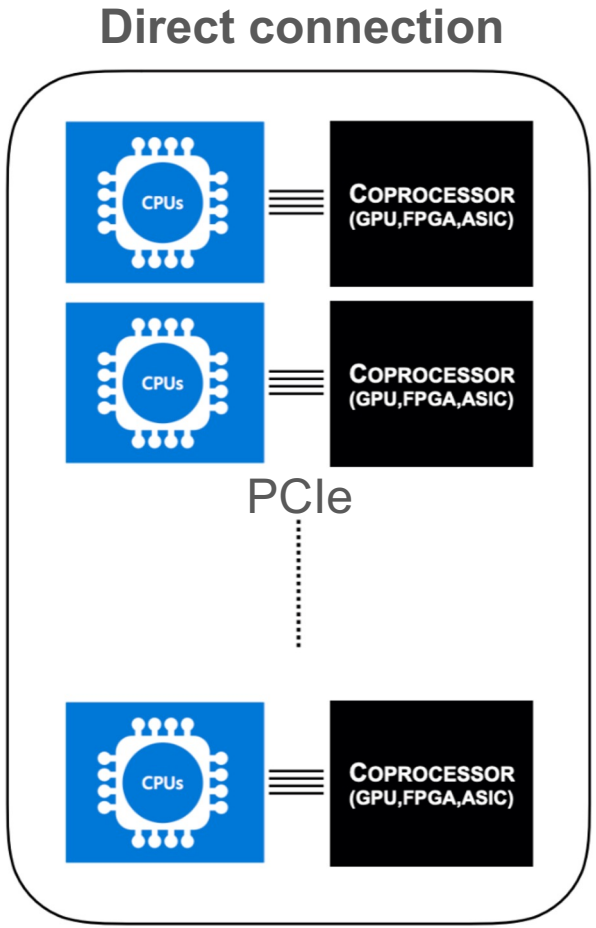


42

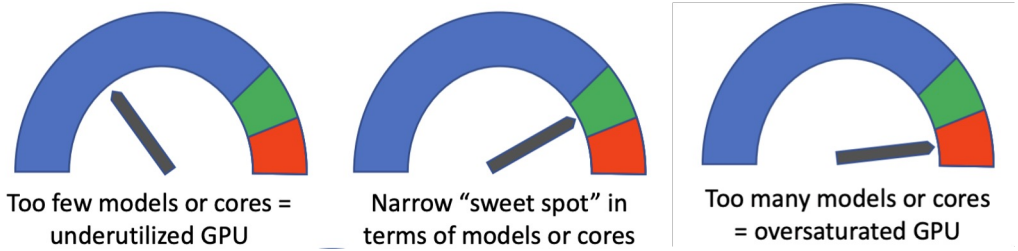
Heterogeneous Computing

The most straightforward way to deploy algorithms on coprocessors is to run on machines with coprocessors

However, **Direct connection** can be inefficient and expensive at scale

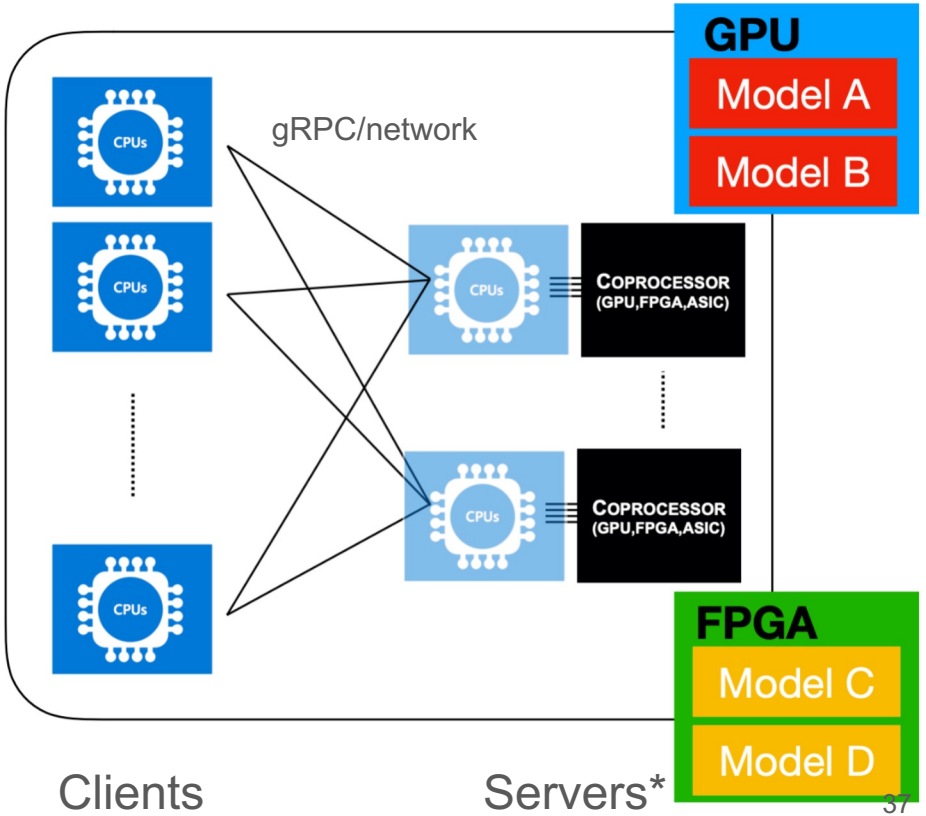


Traditional direct CPU->GPU connection:



As a Service Computing

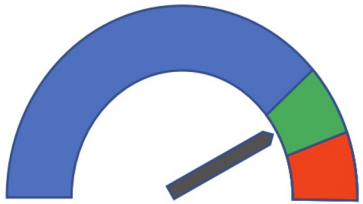
Alternate coprocessor deployment scheme where coprocessor-enabled machines host an inference server and remote jobs send inference requests via network connection



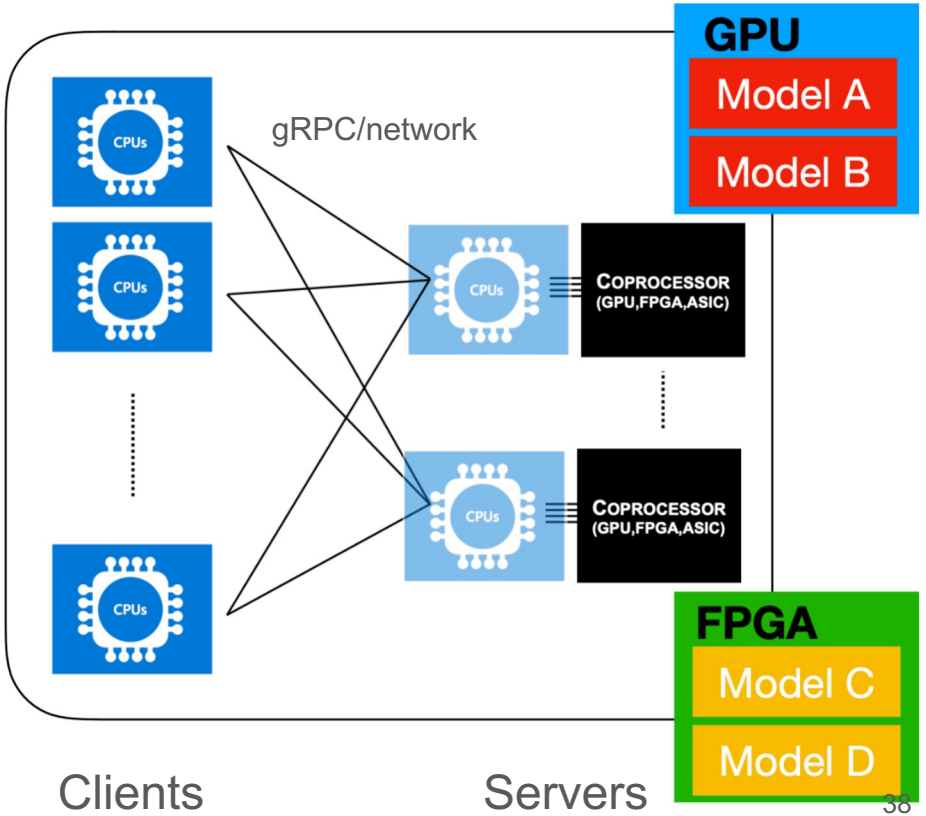
* The servers do not necessarily need to be remote; they can be just next to the client machine.

As a Service Computing

Inference as a Service:



Adjust the number of GPUs per client-CPU core to get as close to the "sweet spot" as possible



Direct

Pros:

- Already have working example

Cons:

- Can be an inefficient use of resources
- Expensive
- Machines without GPUs/FPGAs can't benefit from coprocessors

As a Service Computing

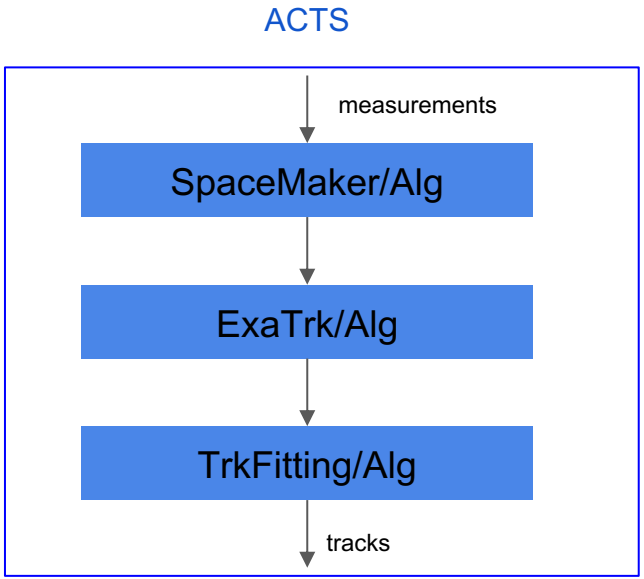
Pros:

- Factorized out the underlying backend implementation
- More straightforward to integrate with the production framework (e.g. Athena)
- Independent of the underlying technology choices and algorithms
- Better scalability and resource utilization (Reduce cost)

Cons:

- Adds complexity

ACTS with GNN (ExaTraX) Plugin (Direct inference)



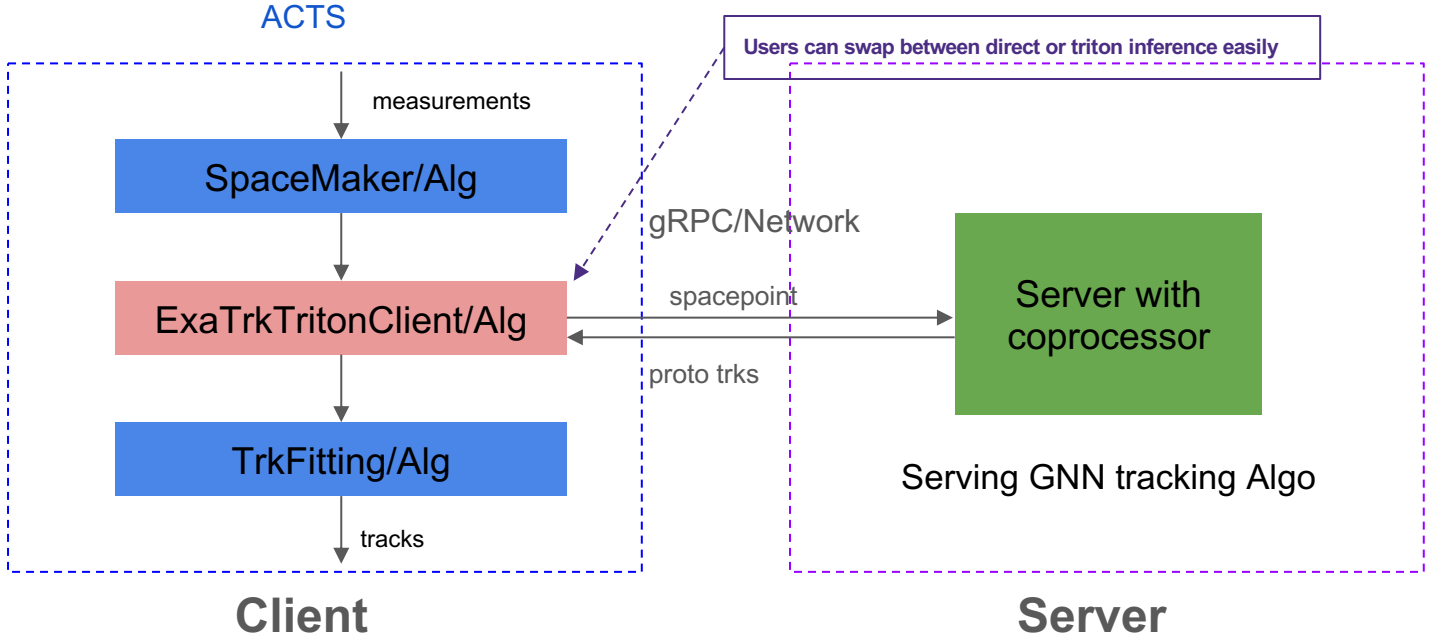
ExaTrk TorchScript implementation done by Benjamin Huth

TrackFinding (ExaTrk) can run locally with CPU/GPU

ACTS TrkFitting still run only on CPU

Integration of the ExaTrkX-as-a-service to ACTS

Client added in ACTS to communicate with Server



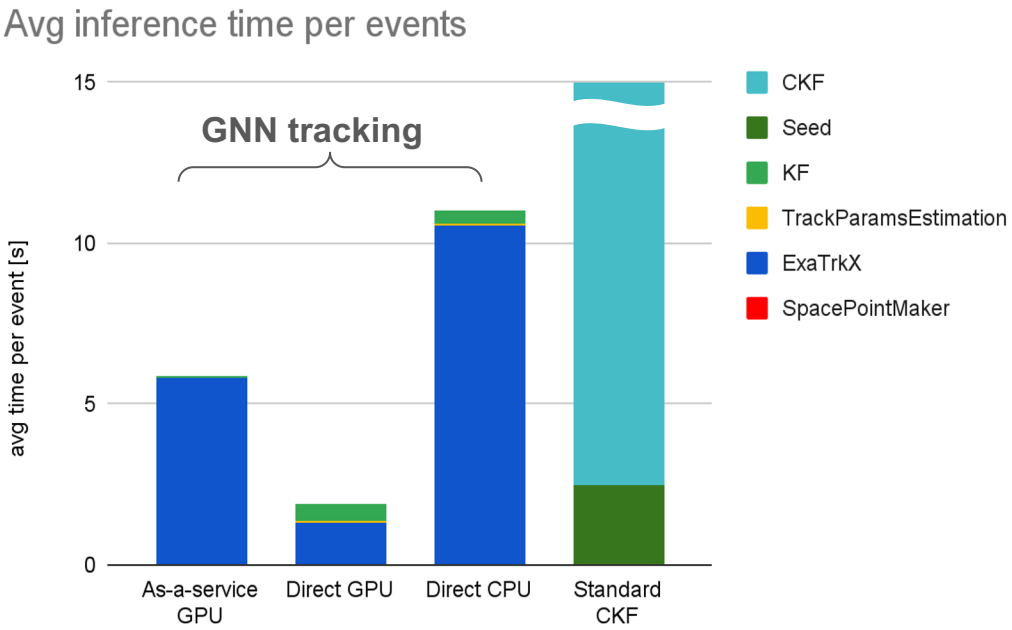
We can offload more algorithm to coprocessor to increase the throughput

Premimary inference timing studies

Direct GPU : 1.6 s

As-a-service GPU: ~5.8 s

- Some overhead in moving to as-a-service.
- Implementation not fully optimized
  - I/O, copying between host & device…etc



ttbar PU200 with OpenDataDetector